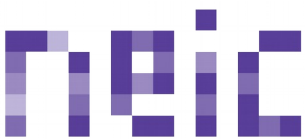
dCache.org

# Walking data with dCache

Tigran Mkrtchyan for dCache people

Sept. 24, Santiago de Compostela






Nordic e-Infrastructure Collaboration



HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

# Today's trip



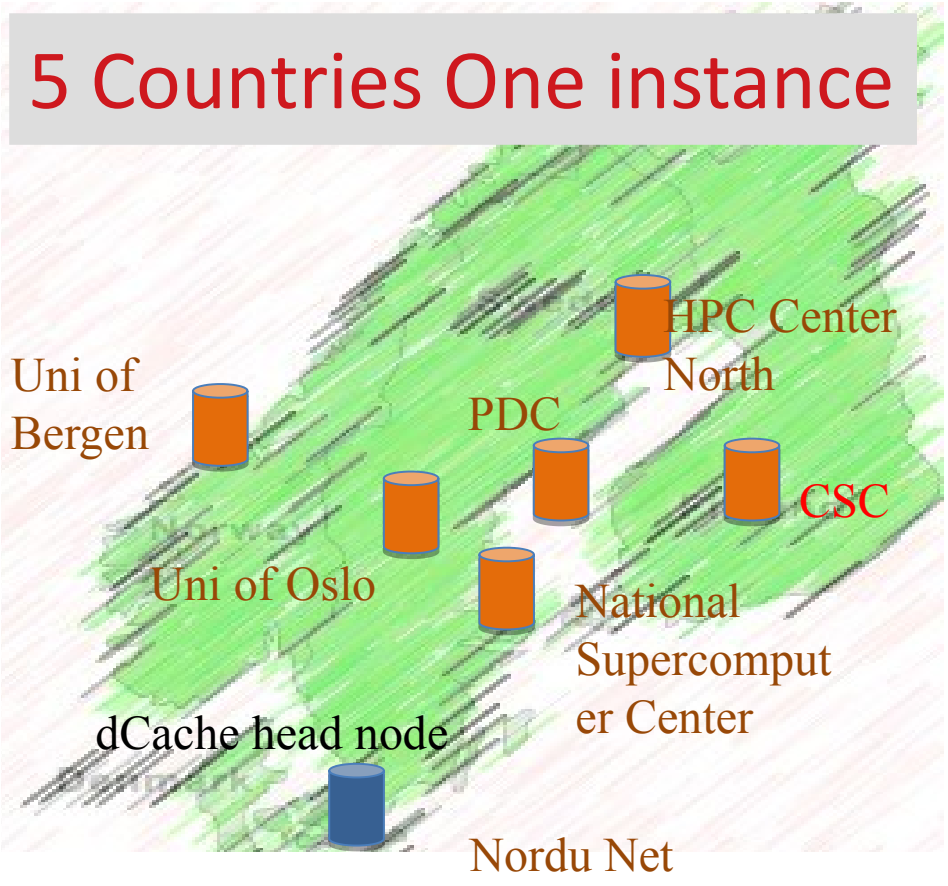
- Introduction to dCache
- How to build a storage system
- Begging data journey
- Scale-up
- Living in distributed world
- Alternative data paths
- Summary

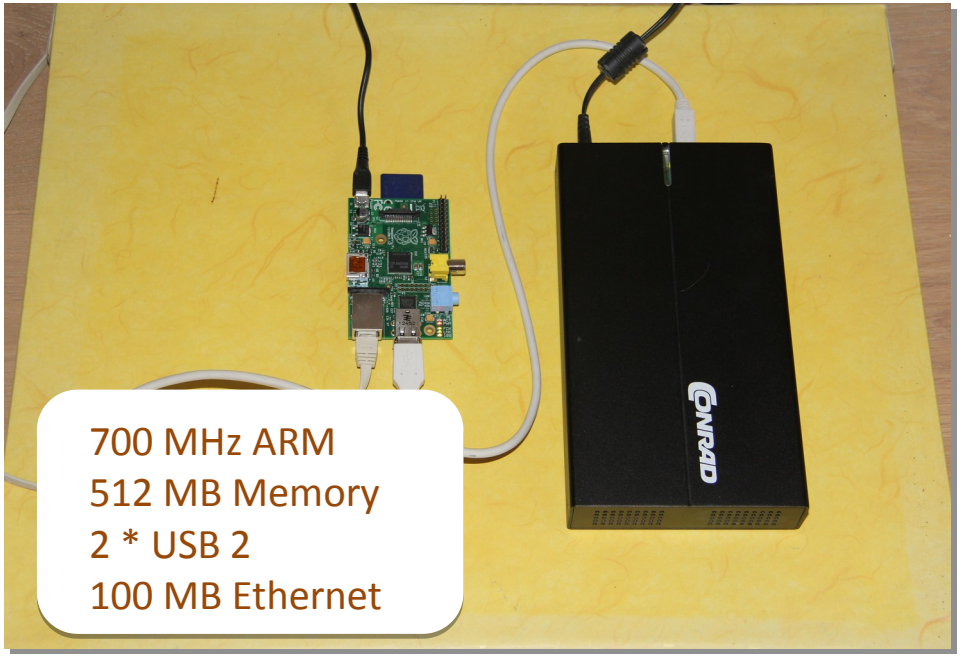
# About dCache

dCache.org

- Joined effort between DESY(2000), FNAL(2001) and NDGF(2006)
- Provides storage solution for scientific data
- Supports standard and HEP specific access protocols
- Supports standard and HEP specific authentication mechanisms
- Developed for HERA and Tevatron, used for LHC and others
  - WLCG, Belle II, LOFAR, CTA, IceCUBE, EU-XFEL, Petra3, DUNE, And many more ...

# From tiny to huge



# dCache: Motivation

dCache.org

- Data never fits into a single server
  - Multiple servers
  - Off-load to tape
- Growing number of client hosts
  - Main frame vs. Linux cluster
- Control over HW/OS selection
  - Better offers
  - Local expertise

# The MISSION

dCache.org

"... to provide a system for storing and retrieving huge amounts of data, distributed among a large number of heterogeneous server nodes, under a single virtual filesystem tree with a variety of standard access methods."

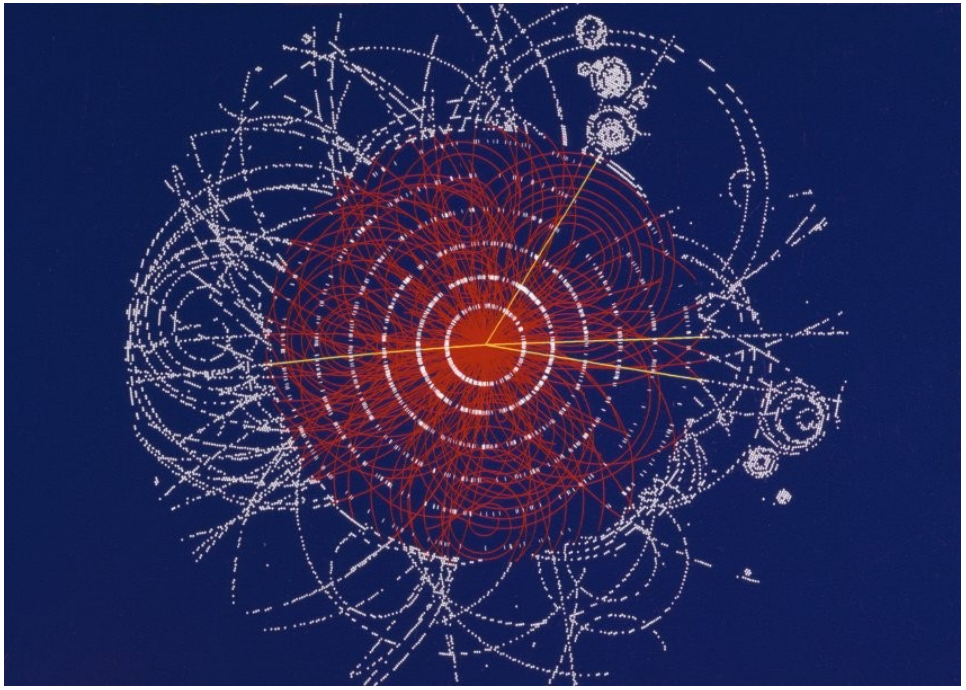
16 Sep. 2000

Michael Ernst, Patrick Fuhrmann, Martin Gasthuber, Rainer Manke

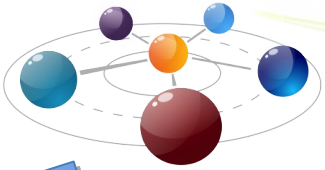
# Scientific data challenges

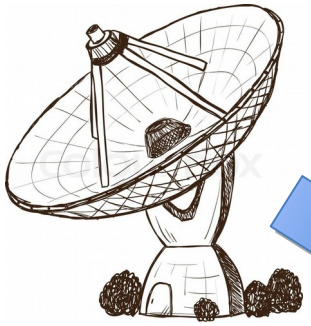


- Volume
- Fast ingest
- Chaotic Access
- Sharing
- Access Control
- Persistence & Long term archival
- Immutability
- Data integrity and protection

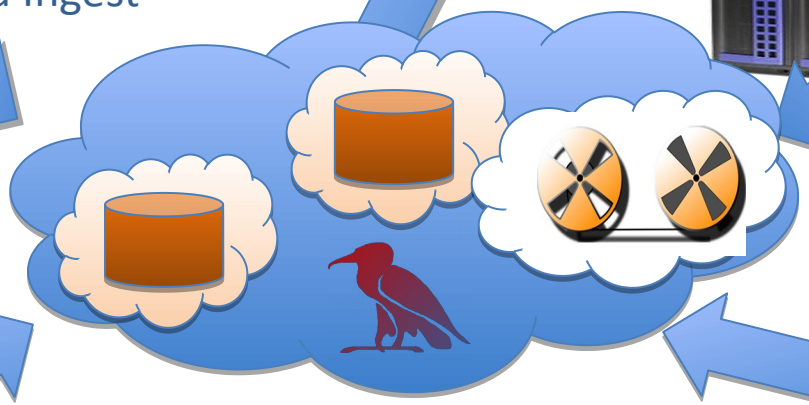


Data management
& workflow control
(Rucio, Kafka, SSE)





High Speed
Data Ingest



Interactive analysis
& Sharing

Fast Analysis
NFS 4.1/pNFS

Wide Area Transfers
(Globus Online, FTS)
by GridFTP, HTTP

# dCache design goals

dCache.org

- Single-rooted namespace, distributed data
- Client talks to namespace for metadata operations only
- Bandwidth and performance grow with number of Pool nodes
- Standard clients (OS native or experiment framework)
- The same data available with any protocol independent from Authentication scheme

# Technical design (1)

dCache.org

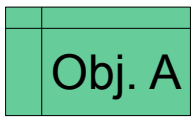
- Set of independent components
  - each component does one thing
- Multiple instances of the same component can be started for horizontal scalability
- Components communicate by sending messages
- *Nowadays such architecture is called* ***microservices with message queue***

# Technical design (2)

dCache.org

- Data and metadata strong separation
- Data servers know only internal unique IDs
  - FileID is independent from the path and remains on rename
  - FileID never re-used
- Data location can be independently added or removed

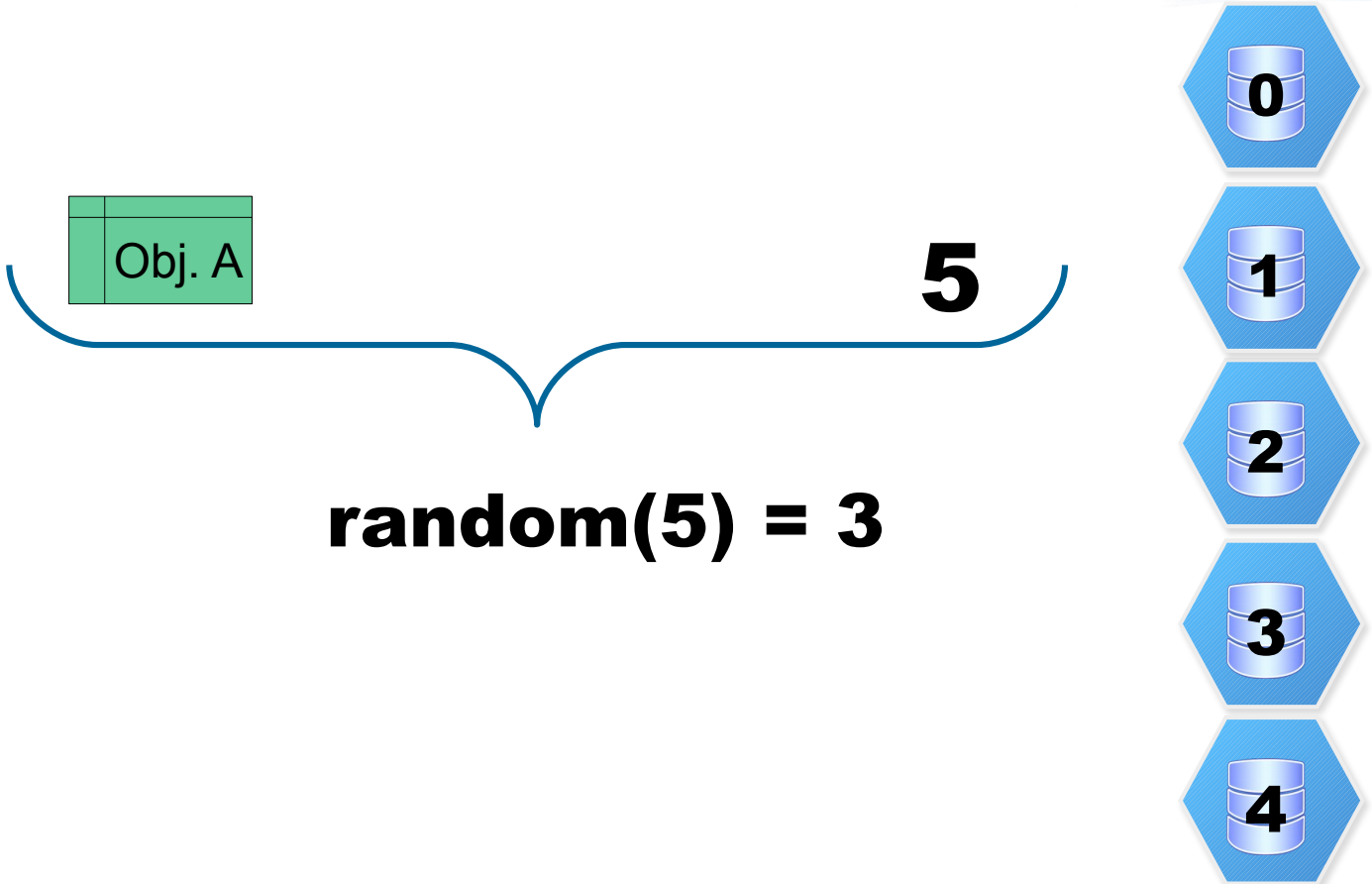
# File placement



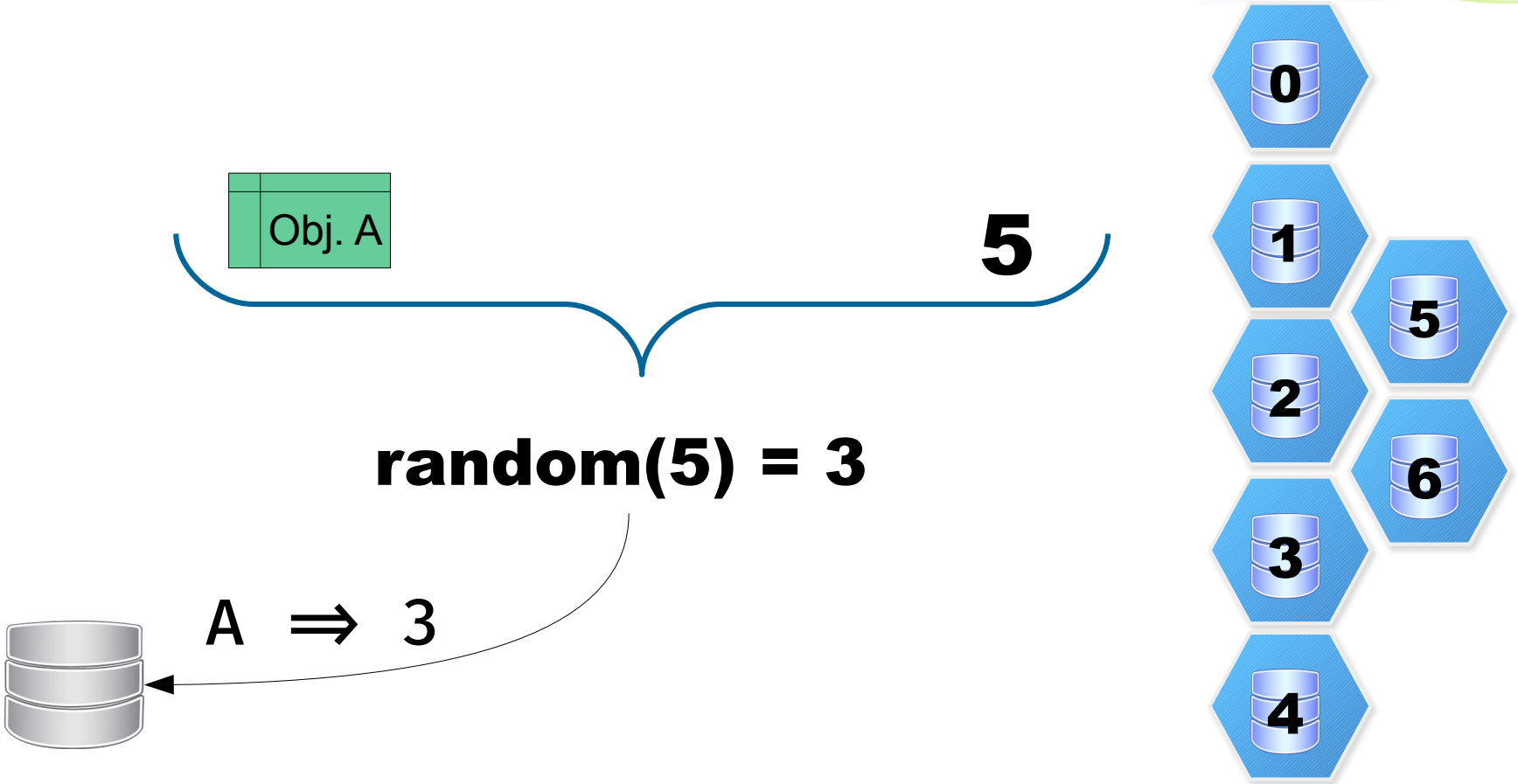

# File placement


**hash(A) % 5 = 3**

# File placement



# File placement



# Tertiary storage support



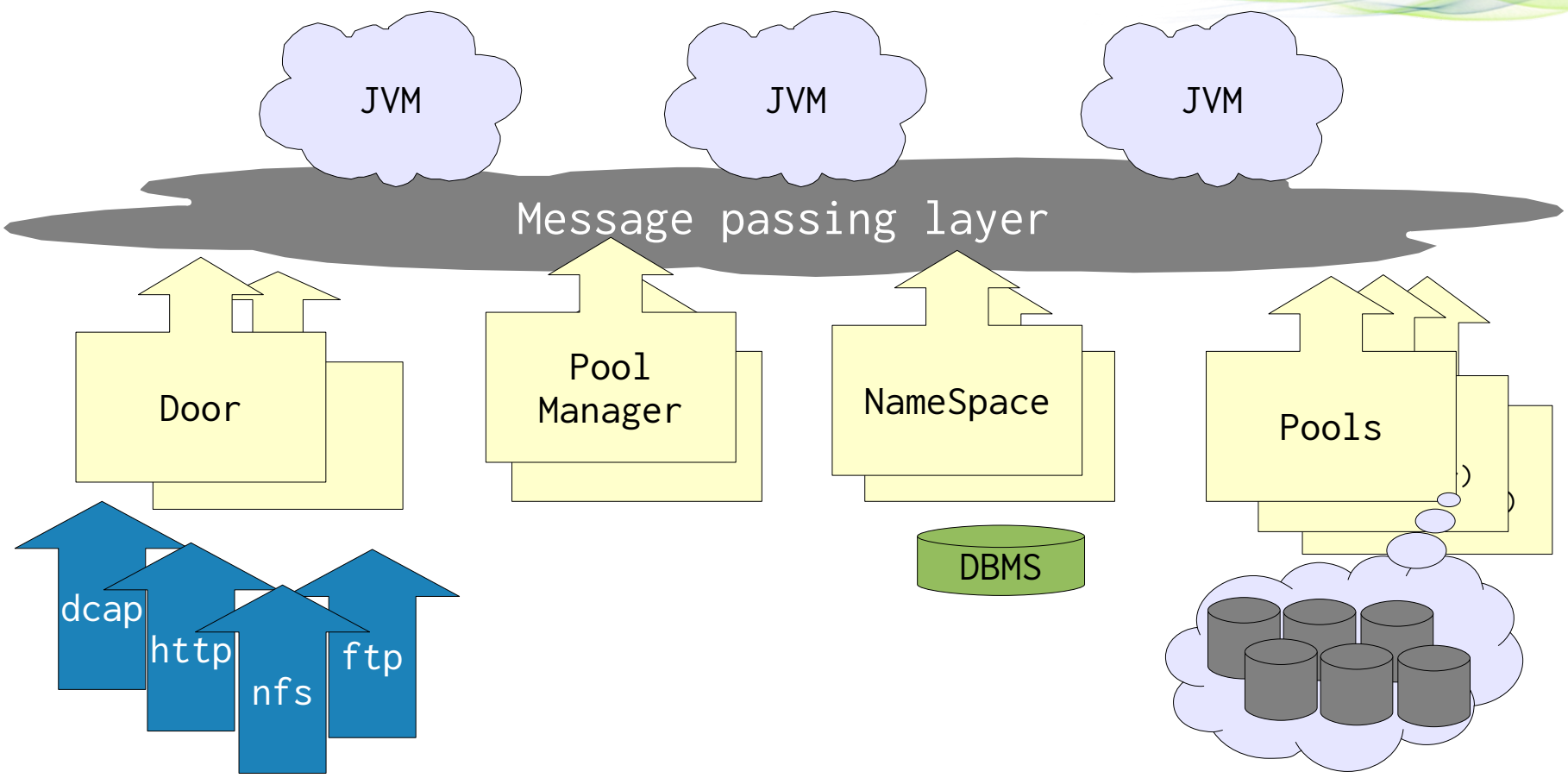
- Native to dCache
  - essential part of original design
- Write-back/read-through -like behavior
  - Transparent for end-users
- Used with a wide variety of HSMs including S3.
- Supports multiple HSMs on the same instance
- Provides full functionality with/without HSM
  - tape and disk-only files can be mixed on the same data server

# Main components

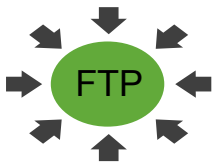
- **DOOR**
  - protocol specific user entry points (NFS, FTP, DCAP, XROOT)
- **POOL**
  - data storage nodes, talk all protocols
- **Namespace**
  - metadata DB, POSIX layer
- **PoolManager**
  - request distribution unit

# dCache on one slide

dCache.org

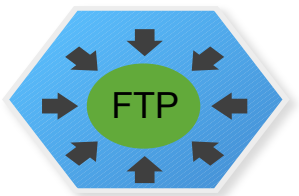


# Minimal Setup



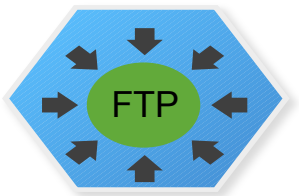



# Minimal Setup





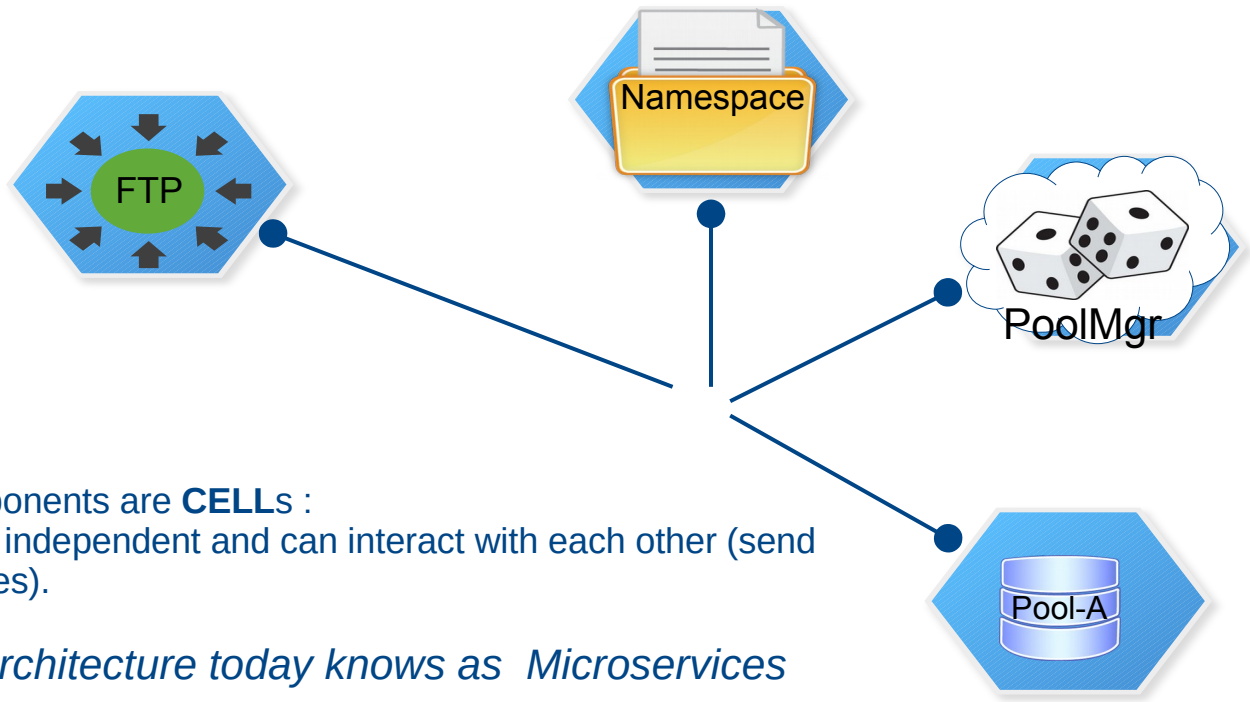

# Minimal Setup






All components are **CELL**s :
they are independent and can interact with each other (send messages).

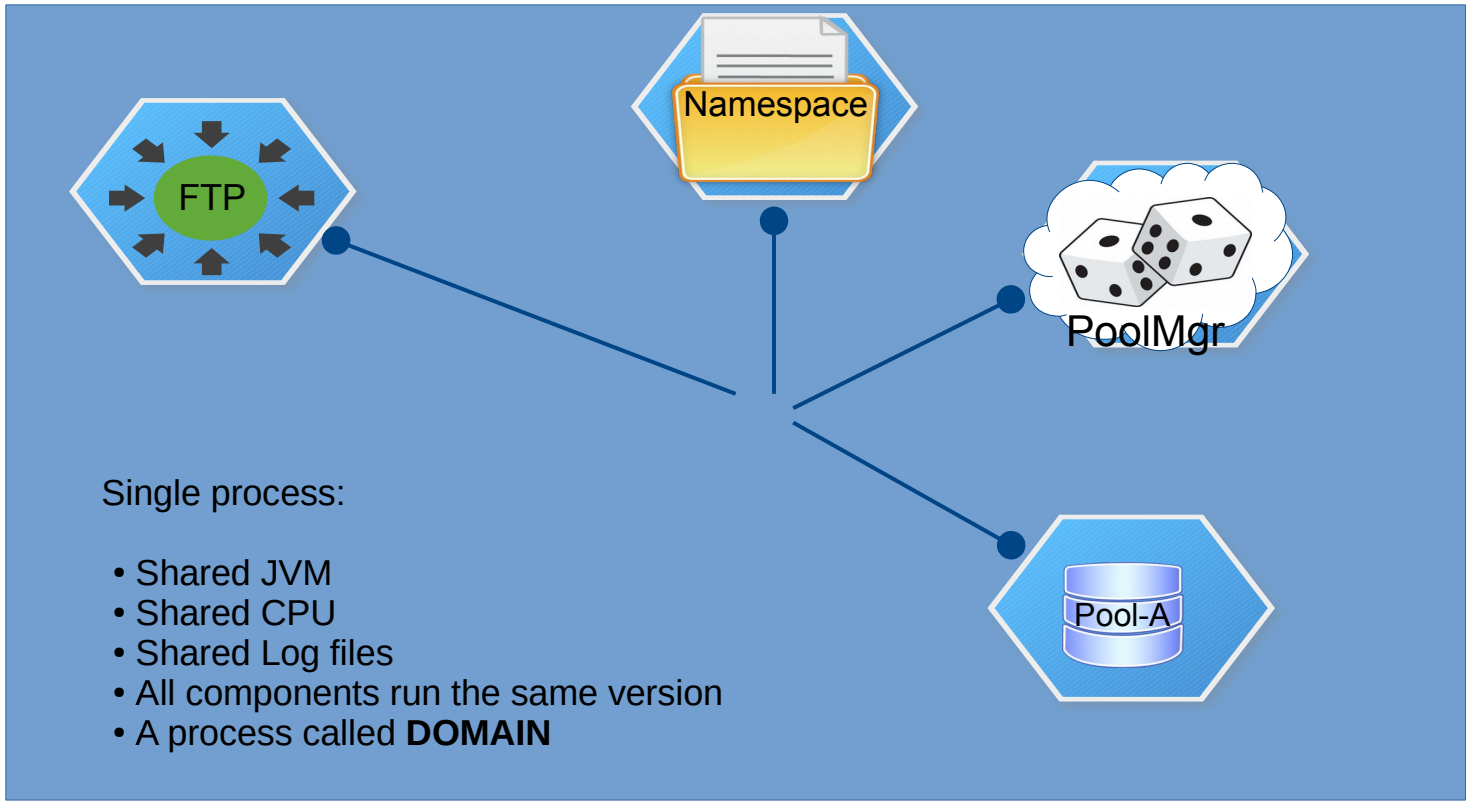
# Minimal Setup



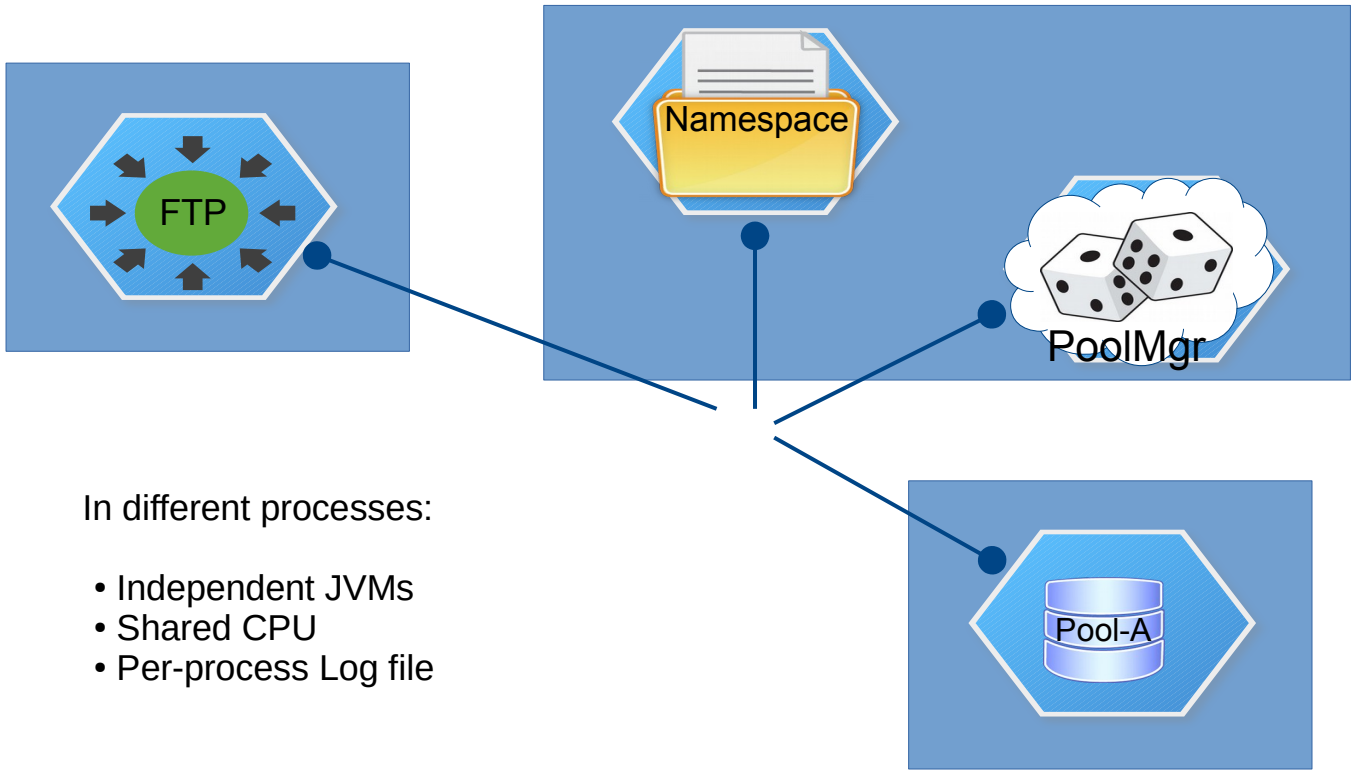

All components are **CELL**s :
they are independent and can interact with each other (send messages).

*Such architecture today knows as  Microservices with message queue.*

# Grouping CELLs



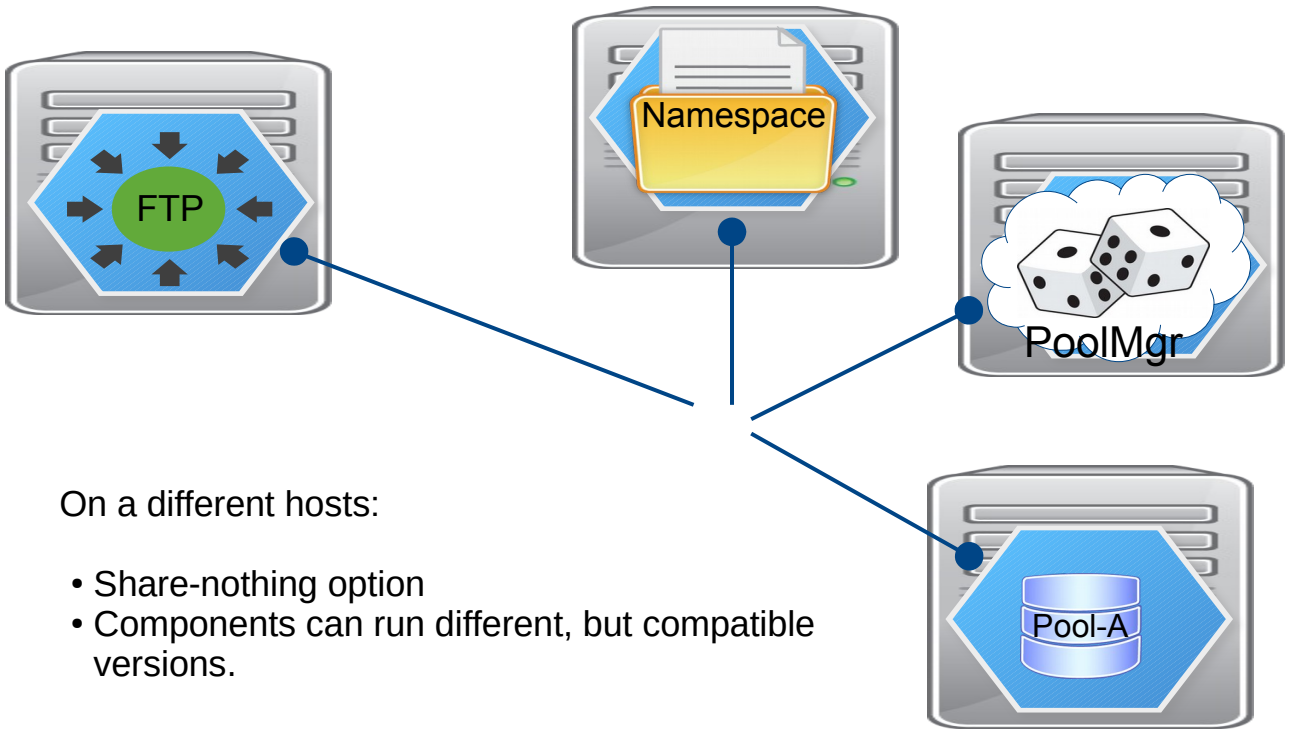
# Grouping CELLs




In different processes:

- Independent JVMs
- Shared CPU
- Per-process Log file

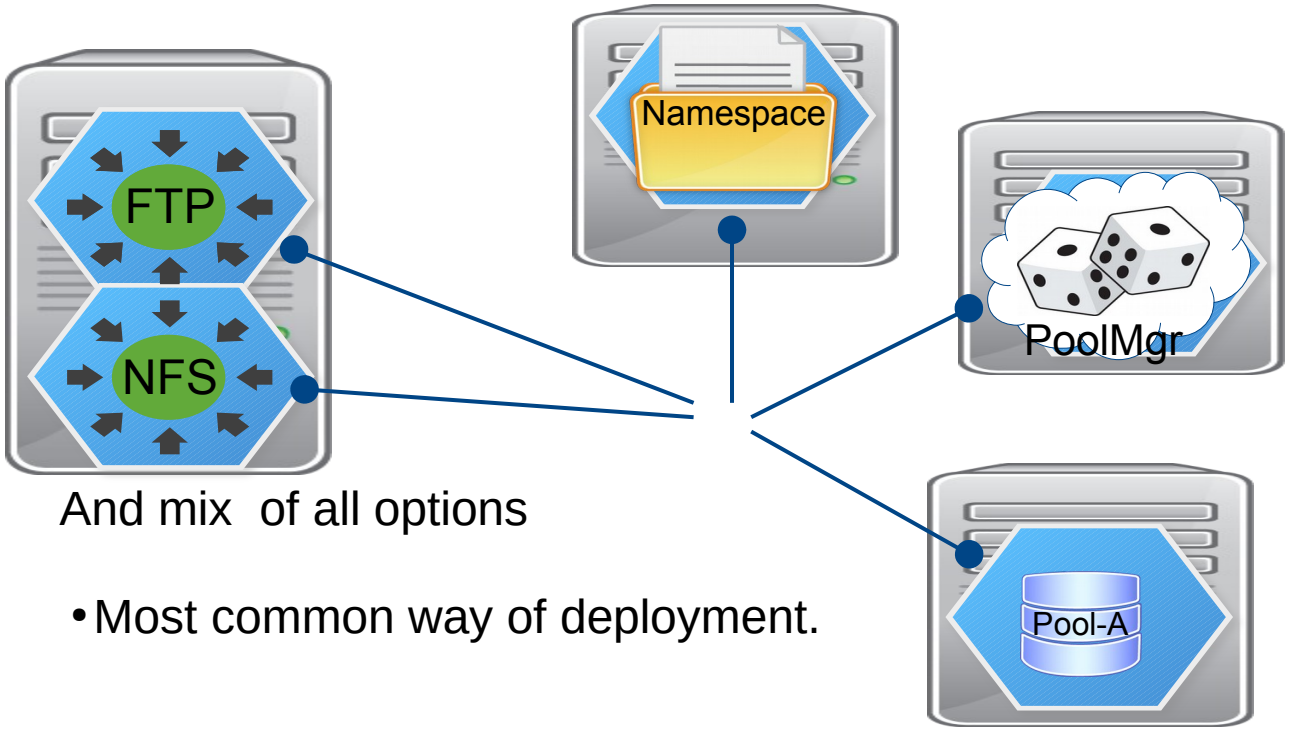
# Grouping CELLs




On a different hosts:

- Share-nothing option
- Components can run different, but compatible versions.

# Grouping CELLs




And mix  of all options

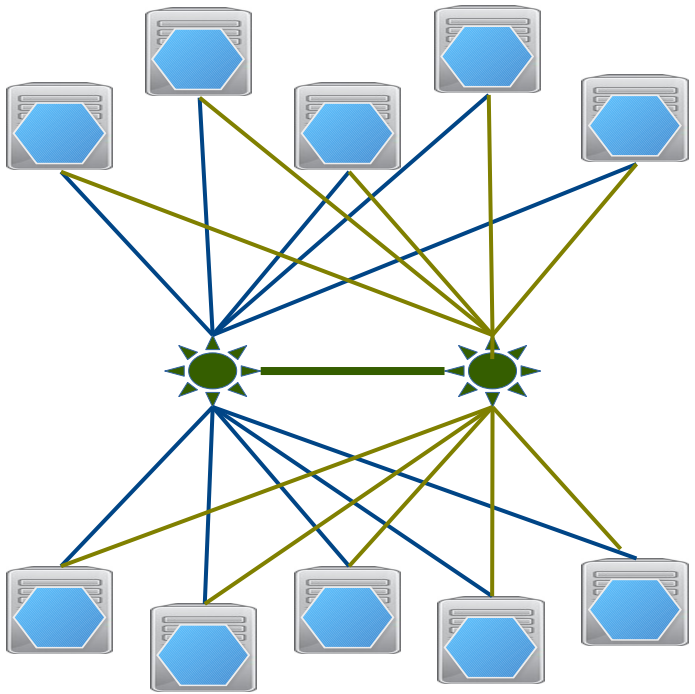
- Most common way of deployment.

# Config

```
[core]
[core/poolmanager]
[core/webdav]

[door]
[door/nfs]
[door/ftp]
```

# Cell messaging





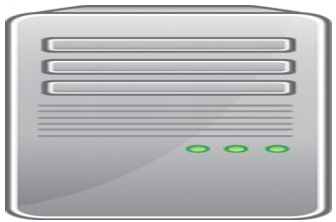
- Star like topology
- Selected node configured as a hub called **CORE** domain
- Other domains called **SATELLITE**
- All communication goes through **CORE** domains
- Multiple **CORE** domains make communication fault tolerant

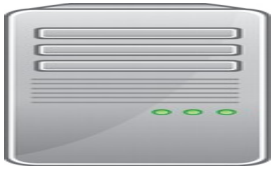
# ZooKeeper as Service discovery



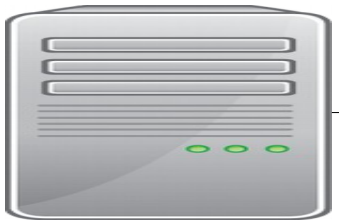
- A central registry of CORE domains
  - Similar to DNS or routing table
- Master election for single-man operations
  - Some actions must be done only once
- Must be deployed as a cluster
  - Built-in fault tolerance

# Service Discovery

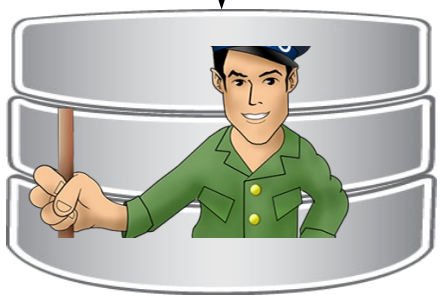




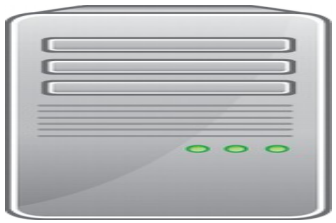
# Service Discovery

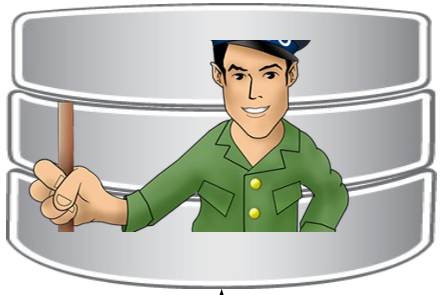


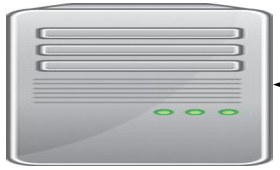
I'm a core Domain, ip:a.b.c.d



# Service Discovery



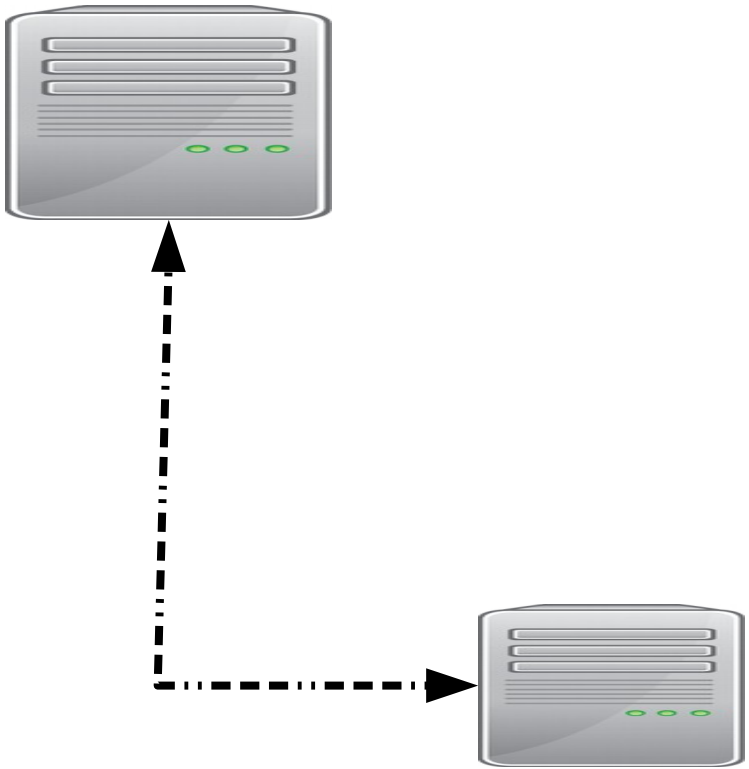




Where are a core Domains?

ip:a.b.c.d

# Service Discovery



# Config

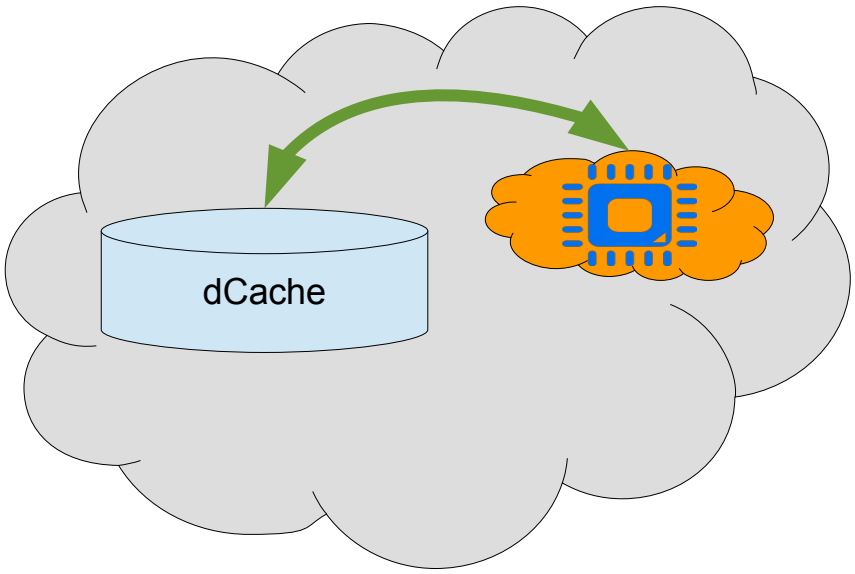
```
dcache.zookeeper.connection = ...
[core]
dcache.broker.scheme = core

[pool]
dcache.broker.scheme = satellite
```

# Fault tolerance



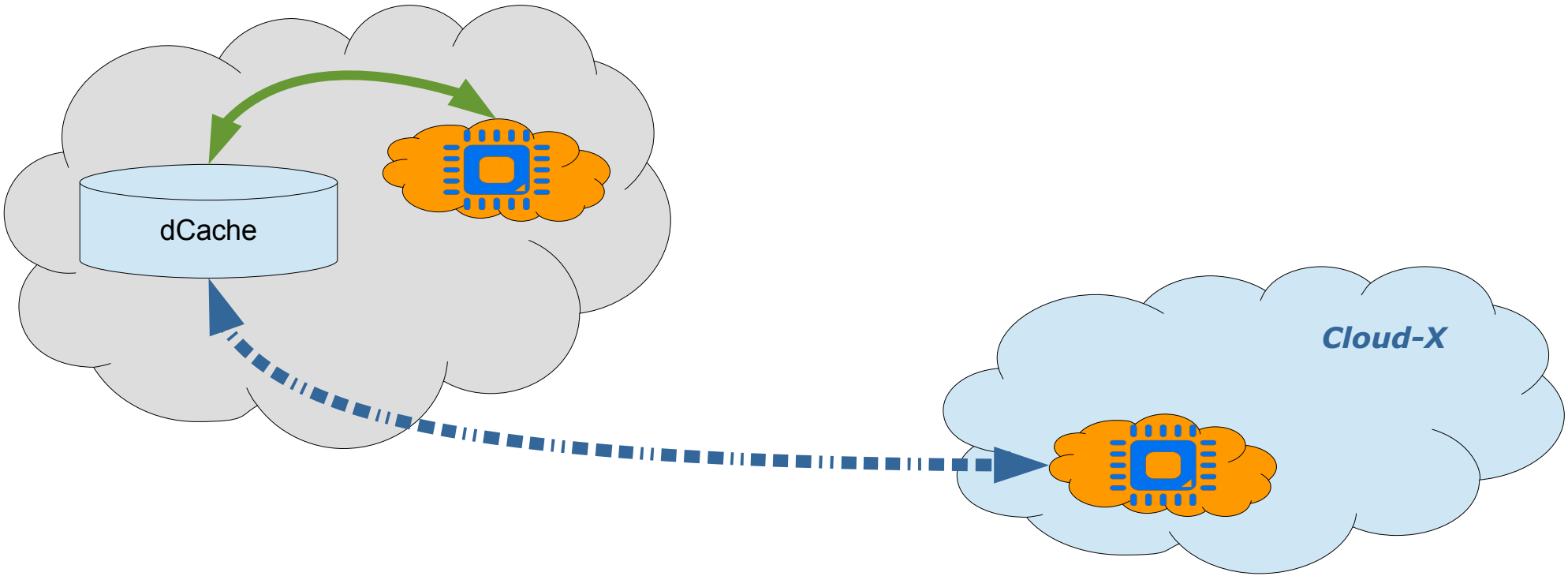
- All core services can run multiple instances
- Door/pool restart handled by clients
  - NFS
  - DCAP
  - XrootD
- Master-slave configuration of namespace database
  - dCache detects which node runs in master mode

# Cloud bursting



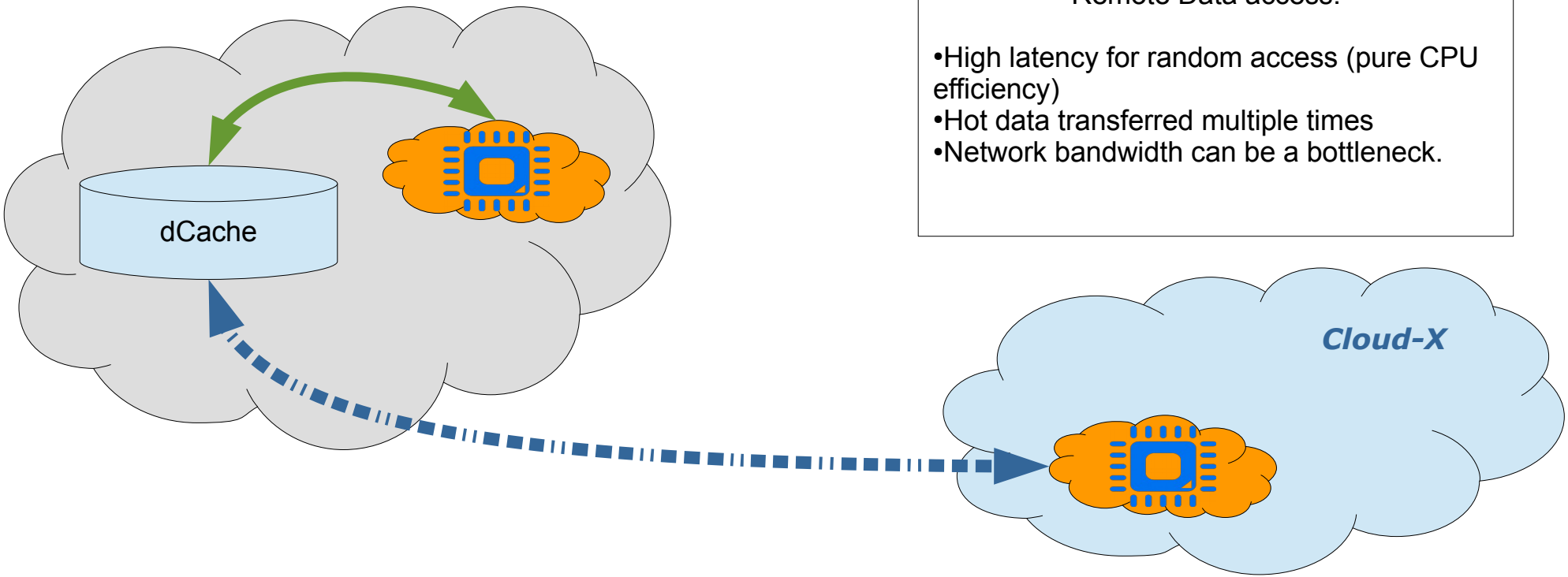

Icon made by Freepik from www.flaticon.com

# Cloud bursting




Icon made by Freepik from www.flaticon.com

# Cloud bursting

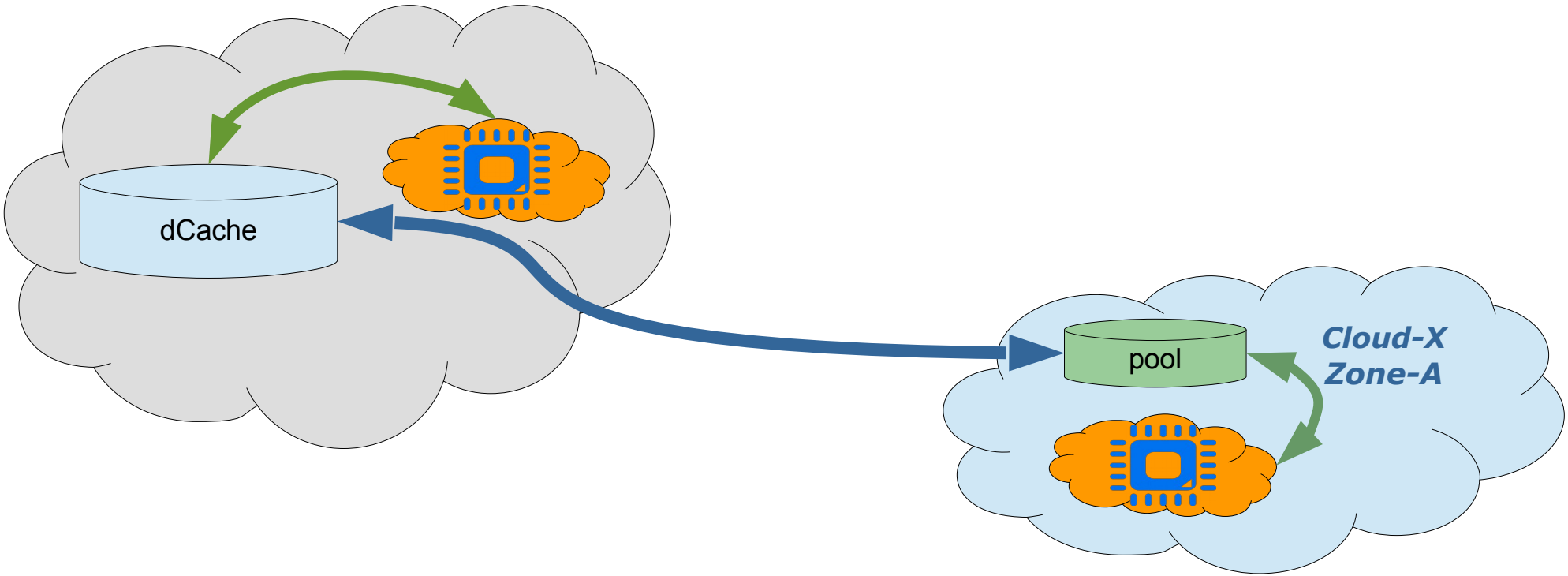



Remote Data access:

- High latency for random access (pure CPU efficiency)
- Hot data transferred multiple times
- Network bandwidth can be a bottleneck.

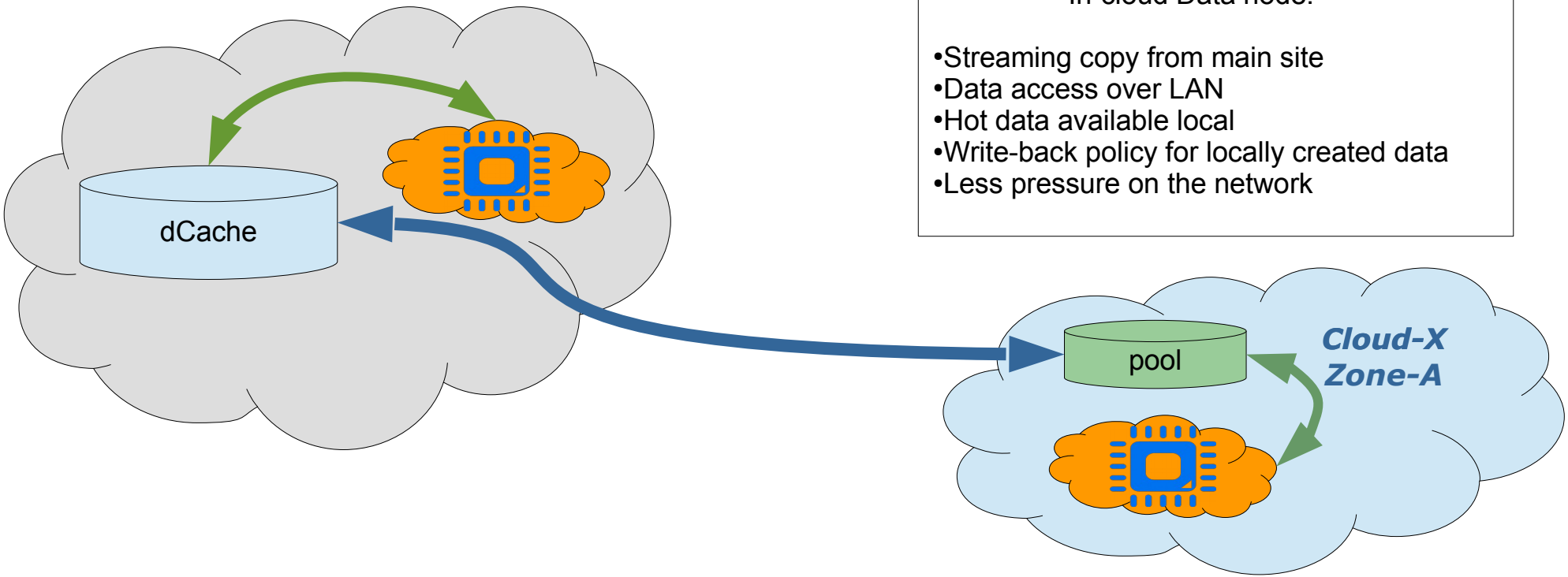
Icon made by Freepik from www.flaticon.com

# Cloud bursting




Icon made by Freepik from www.flaticon.com

# Cloud bursting

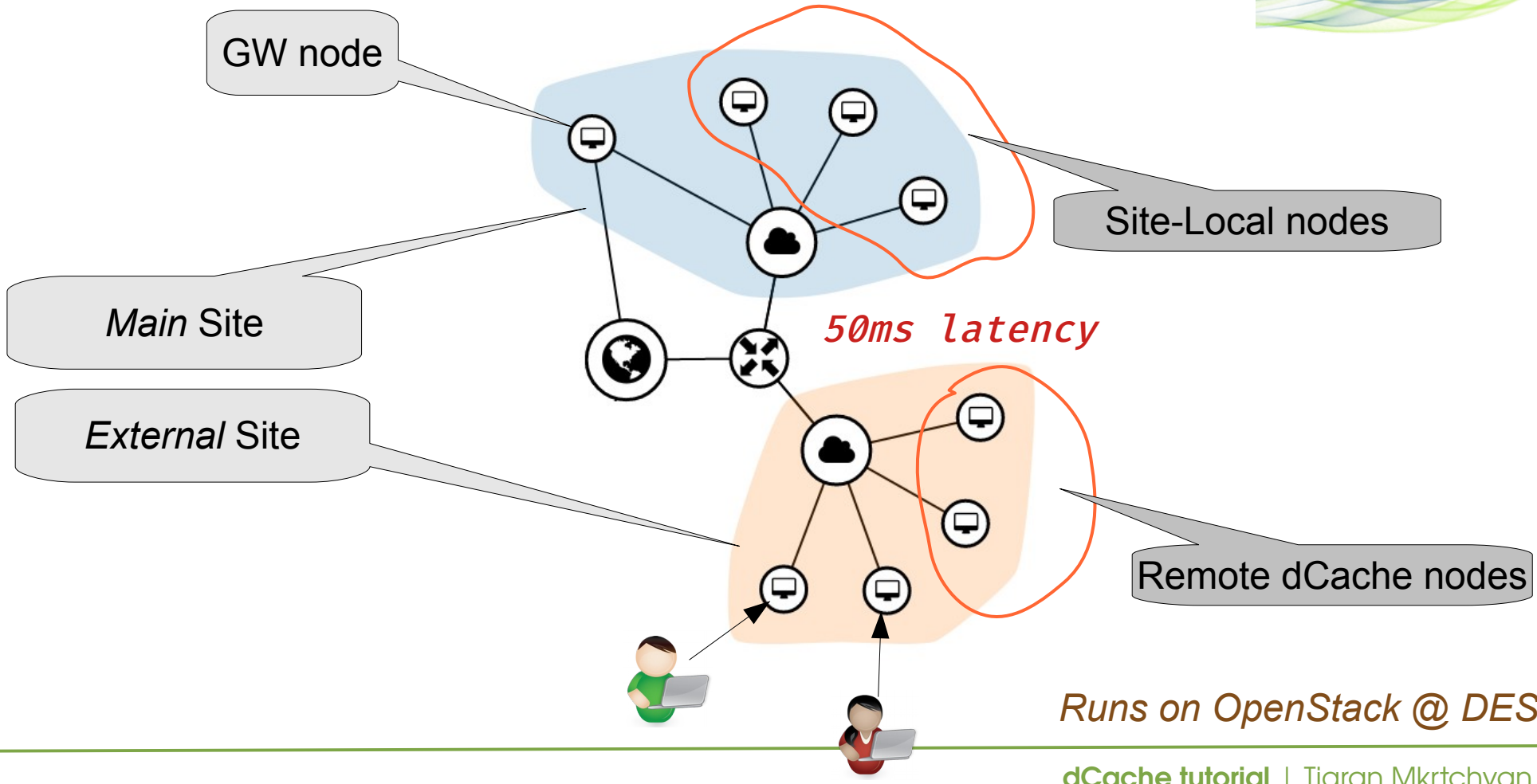



In-cloud Data node:

- Streaming copy from main site
- Data access over LAN
- Hot data available local
- Write-back policy for locally created data
- Less pressure on the network

Icon made by Freepik from www.flaticon.com

# Demo setup

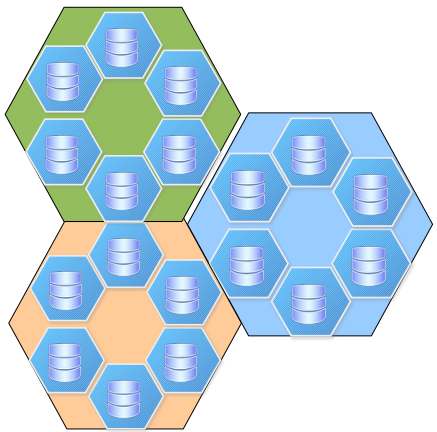



*Runs on OpenStack @ DESY*

## Request Parameters

- Protocol
- Path (store)
- Client IP

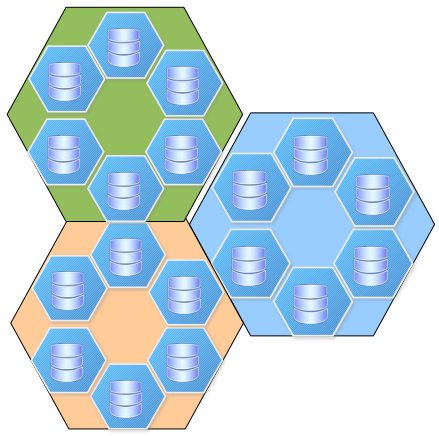
## Pool Groups



## Request Parameters

- Protocol
- Path (store)
- Client IP


## Pool Groups



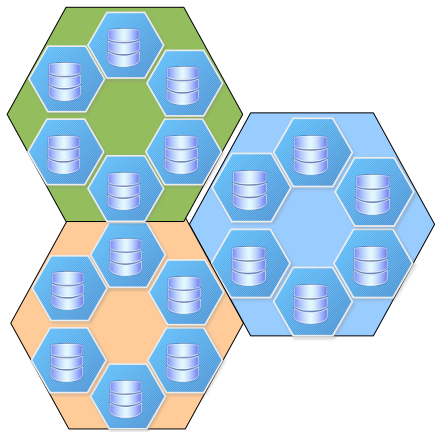
## Request Parameters

- Protocol
- Path (store)
- Client IP

**Link**

| Operation | Priority |
|---|---|
| READ | 10 |
| WRITE | 10 |
| CACHE | 5 |
| P2P | 1 |

## Pool Groups

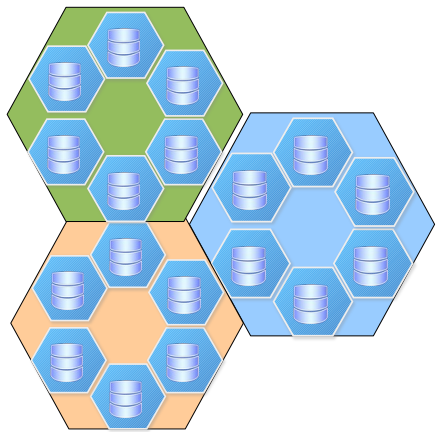


## Request Parameters

- Protocol
- Path (store)
- Client IP

**Link**

## Pool Groups



| Operation | Priority |
|---|---|
| READ | 10 |
| WRITE | 10 |
| CACHE | 5 |
| P2P | 1 |

When multiple ***links*** match,then a link with highest priority is selected!

# Example: (simplified 😁)

dCache.org

```
... falback-pools
... desy-pools
... extern-pools -dynamic -tags=zone=extern
... desy-net 131.169.0.0/16 2001:638:700::/48
... extern-net 159.93.0.0/16

psu create link desy-link desy-net any-protocol desy-pools
psu set link desy-link -readpref=10 -writepref=10

psu create link extern-link extern-net any-protocol extern-pools
psu set link extern-link -readpref=10 -writepref=10

psu create link default-link any-net any-protocol fallback-pools
psu set link default-link -readpref=1 -writepref=1
```

# Config

```
dcache.zone=extern

[${host.name}]
[${host.name}/pool]

pool.name=pool-${host.name}
pool.path=/pool
```

# dCache and CAP

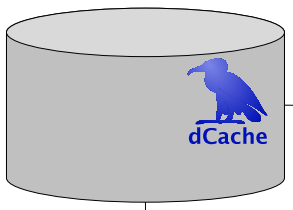
- dCache provides consistency over availability.
- All client will see the same data at the same time.
- A timeout or error will be returned, if consistency can't be guaranteed.

# Storage events in dCache

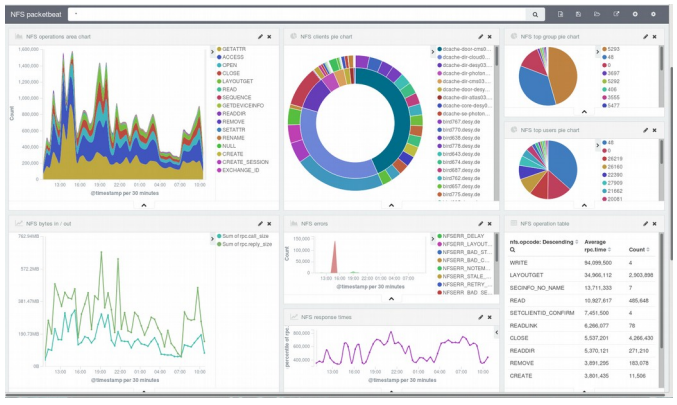
dCache.org

- Kafka stream
  - Producer-consumer model
  - Kafka consumer is required
  - global events
  - Consumer keeps track of the last seen event
  - Integration with other tools (Spark, ELK, ...)
- Server-Send Events (SSE)
  - Producer-consumer model
  - HTTP connection "for receiving push notifications from a server"
  - User specific event stream
  - Client keeps track of the "Last-Event-ID"

# Storage events

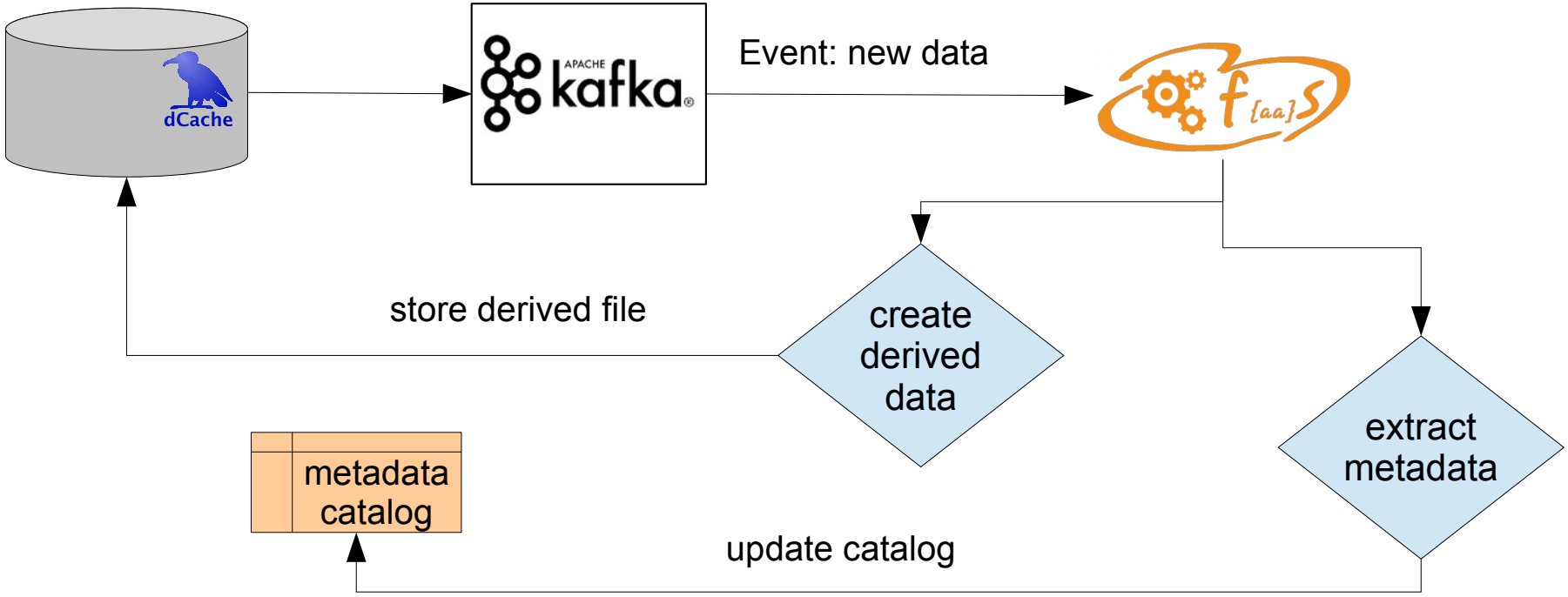






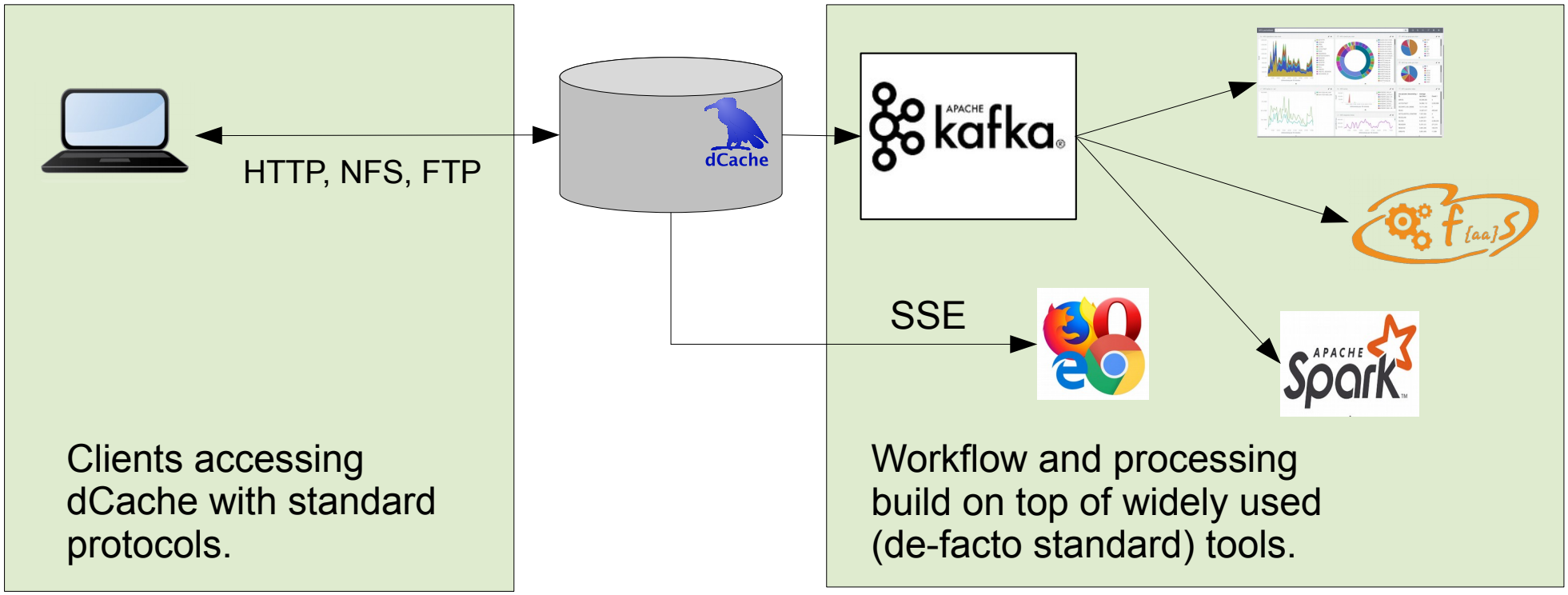
SSE

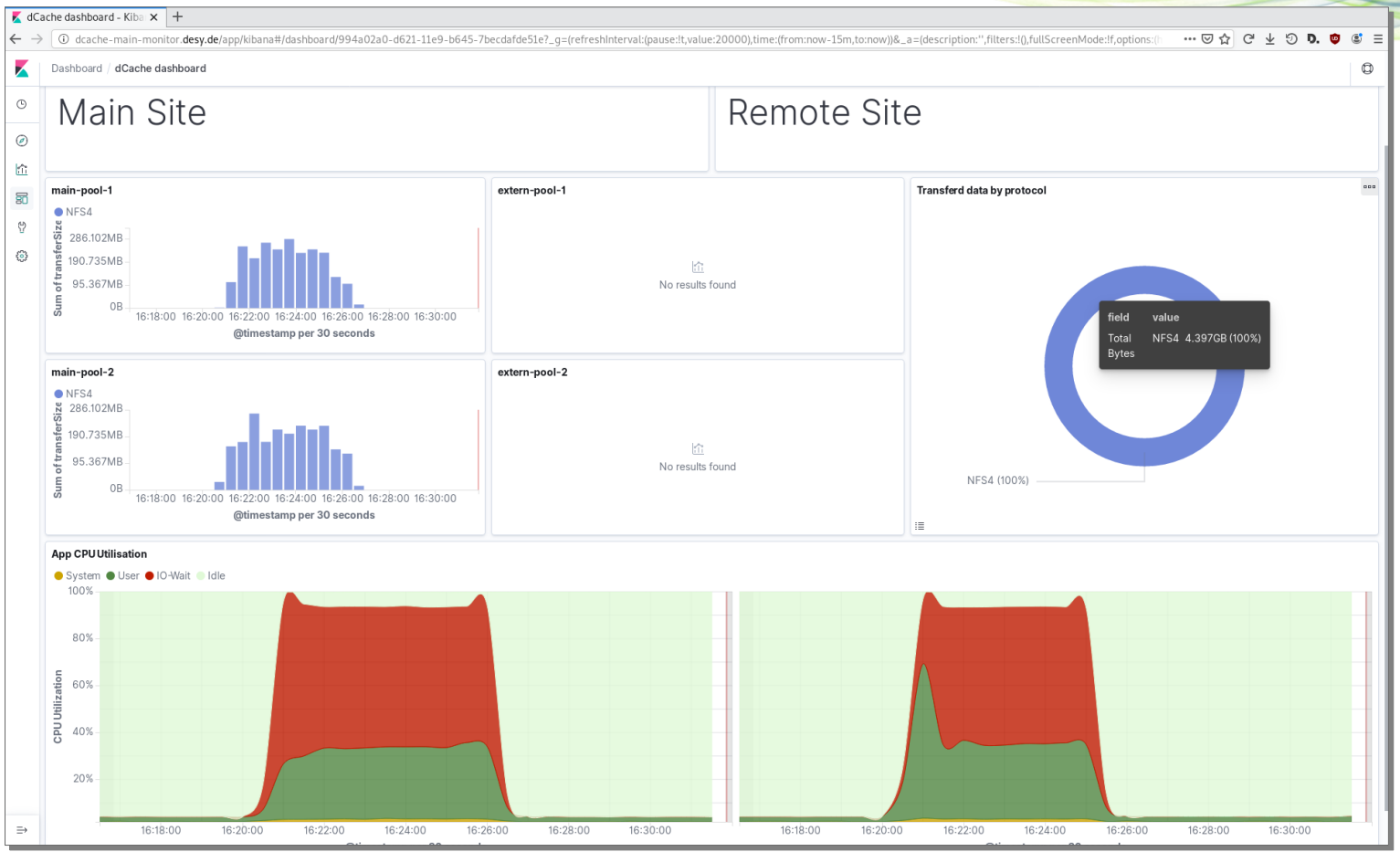

# Workflow control




by Michael Schuh (Data Ingesting and Processing in the EOSC)

# Standards everywhere...




dCache dashboard - Kiba
dcache-main-monitor.desy.de/app/kibana#/dashboard/994a02a0-d621-11e9-b645-7becdafde51e?_g=(refreshInterval:(pause:!t,value:20000),time:(from:now-15m,to:now))&_a=(description:'',filters:!(),fullScreenMode:!f,options:(
Dashboard / dCache dashboard
Main Site
Remote Site
main-pool-1
NFS4
Sum of transferSize
286.102MB
190.735MB
95.367MB
0B
16:18:00 16:20:00 16:22:00 16:24:00 16:26:00 16:28:00 16:30:00
@timestamp per 30 seconds
extern-pool-1
No results found
Transferd data by protocol
field value
Total Bytes NFS4 4.397GB (100%)
NFS4 (100%)
main-pool-2
NFS4
Sum of transferSize
286.102MB
190.735MB
95.367MB
0B
16:18:00 16:20:00 16:22:00 16:24:00 16:26:00 16:28:00 16:30:00
@timestamp per 30 seconds
extern-pool-2
No results found
App CPU Utilisation
System User IO-Wait Idle
CPU Utilization
100%
80%
60%
40%
20%
16:18:00 16:20:00 16:22:00 16:24:00 16:26:00 16:28:00 16:30:00
16:18:00 16:20:00 16:22:00 16:24:00 16:26:00 16:28:00 16:30:00

# Re-cap



- dCache is a widely used storage system
- Flexible architecture allows to deploy system that match your needs
- Storage capacity can be dynamically added
- Data placement rules can distribute data according network topology
- Efficient use of external resources

# Thank You!

*More info:*

*https://www.dcache.org*

*To steal and contribute:*

*https://github.com/dCache/dcache*

*Demo:*

*https://github.com/dCache/dcache-on-docker*

*Help and support:*

*support*  *dcache.org*